# Processing conversion and parallel control platform: a parallel approach to serial hydrodynamic simulators for complex hydrodynamic simulations

Yizi Shang, Yanxiang Guo, Ling Shang, Yuntao Ye, Ronghua Liu and Guangqian Wang

## ABSTRACT

In this paper, a processing conversion and parallel control platform (PCsP) is proposed for transitioning serial hydrodynamic simulators to a cluster-computing system. We previously undertook efforts to promote the research and development of this type of platform and to demonstrate and commercialize it. Our PCsP provide distributed and parallel patterns, a centralized architecture, and user support. To validate our employed methodology and highlight its simplicity, we adopted the technology in various applications based on multi-grid algorithms. The methodology was shown to be reliable and feasible across computational domains, partitioning strategies, and multi-grid codes. Furthermore, its effectiveness was demonstrated using a complex engineering case in addition to code based on slightly less complex mathematical models. Eventual transition to a cluster-computing system will require further investigation of the impact of different model combinations on calculation accuracy, efficiency of operating models, and PCsP functional development.

**Key words** | cluster computing, distributed and parallel patterns, hydrodynamic simulator, multi-grid algorithm

**Yizi Shang**
**Yuntao Ye**
**Ronghua Liu**
State Key Laboratory of Simulation and Regulation of Water Cycle in River Basin,
China Institute of Water Resources and Hydropower Research,
Beijing 100038,
China

**Yizi Shang**
**Yanxiang Guo**
**Yuntao Ye**
**Ronghua Liu**
**Guangqian Wang**
State Key Laboratory of Hydroscience and Engineering, Tsinghua University,
Beijing 100084,
China

**Ling Shang** (corresponding author)
School of Information Engineering,
North China University of Water Conservancy and Electric Power, Zhengzhou,
China
and
Dahe Cluture and Finance Group,
Nanjing,
China
E-mail: *ling.shang.dahe@gmail.com*

## INTRODUCTION

Owing to improvements in parallel computing and networking as well as the increasing availability of computer clusters, the development of an approach that effectively utilizes computing power has been a major objective for hydraulic engineers since the mid-2000s. This is especially applicable to hydrodynamics, in which large sets of equations must be simultaneously solved (Afshari *et al.* 2008). Parallel computing can leverage the problem-solving ability of computer clusters and requires parallel algorithms and a supporting framework (Holmes *et al.* 2011). The parallel implementation of an algorithm is traditionally based on topologies for structuring knowledge, thereby facilitating the efficient creation of complex relationships among data by means of specific languages, including different levels of abstraction (e.g. PRAM, BSP, and LogP) (Chui *et al.* 2009). Despite its parallel processing ability, such an implementation faces an underlying problem in that distributed and parallel programming mainly focus on coding topologies and complex relationships (Márquez *et al.* 2011). This makes parallel applications inaccessible and unmanageable from the perspective of hydraulic engineers, who are rarely experts in topological representations and languages (Chen & Lin 2008). Therefore, most specialized programs that have been developed by hydraulic engineers are limited to serial computation.

doi: 10.2166/hydro.2016.204

The need to reuse these types of programs to develop parallel applications has motivated recent research efforts. However, these efforts face two major hurdles (Jagers 2010). First, the programs generally provide very limited features because they are developed primarily to prove some concepts or investigate certain optimizations. Thus, they are unlikely to provide the required power and utility compared with their corresponding applications. In addition, the lack of a supporting framework means that these programs must be constructed from scratch and with multiple programming languages. Therefore, an urgent need exists for an innovative approach and related techniques for converting existing serial programs into parallel applications without modifying the source code and without the need for expert knowledge of topological representations and languages.

This paper relays our experience of developing such a cluster-computing environment for available facilities. The tangible contribution of this project is a processing conversion and parallel control platform (PCsP), which exclusively converts a program into a module for parallel applications by inserting a procedural function into the program loops. Accordingly, serial programs can be processed in parallel on a computer cluster. The objective of this approach is to improve serial programs through a simple parallel design scheme. This approach shows high performance and effectively speeds up multi-core processors. Moreover, numerical results obtained with this method demonstrate its accuracy with respect to the computational domain (in particular, when considering hybrid computing). Furthermore, the effectiveness of the approach was demonstrated in a practical engineering case (Wang *et al.* 2011a) and with code based on relatively less complex mathematical models (Shang *et al.* 2011a, 2011b).

Ongoing projects have shifted our attention to hydrodynamic simulators developed on the basis of multi-grid algorithms. Our approach can be used to parallelize heterogeneous code from a wide range of time-dependent models. In this context, we herein detail the PCsP's development and show how all of the content in a serial program can be encapsulated in a parallel module on the basis of a geometric partitioning strategy (Fialko 2010).

The remainder of this paper is organized as follows. The next section briefly reviews related studies, followed by a section describing the architecture of the proposed PCsP. Implementation issues are discussed in the following section, and the final section summarizes our findings and concludes the paper.

## RELATED WORK

The practical approaches that researchers have used when devising parallel applications are based on providing either domain-specific tools or general-purpose solutions. The domain-specific approach is to supply users with standard application programming interfaces (APIs), through which a serial program can be executed in a distributed and coordinated manner (Shang *et al.* 2007a). To this end, a developer must, in principle, indicate which parts of the program will benefit from being parallelized. This is accomplished by inserting sequential code that implements appropriate calls to the APIs. Through the provision of APIs for executing different components of a program, the program is parallelized by dividing it into a number of distributed components that communicate through exchanged messages (Shang *et al.* 2011a, 2011b). A message passing interface (MPI) (Shang *et al.* 2007b) and parallel virtual machine (PVM) (Gregersen *et al.* 2007) are two well-established parallel computing environments. Object-oriented methods are used to develop APIs based on an MPI or PVM. These object-oriented methods mitigate the complexity inherent to writing parallel applications by encapsulating common distributed and parallel patterns in an intuitive API. A brief review of the development of these APIs can be found in the literature (Moore & Tindall 2005; Bugaets 2014). However, this approach has attracted significant criticism because MPIs and PVMs are basically low-level parallelization tools that require users to have substantial code-specific knowledge of both parallel programming and distributed deployment. Therefore, considerable manual effort is required whenever a new application is considered.

On the other hand, the general-purpose approach tends to relieve users of parallelization and deployment tasks as much as possible by raising the level of abstraction to users in the APIs. In general, this approach adopts a centralized architecture to provide a high-level interface for accessing and manipulating components of a program by using an API that conforms to component object model (COM) automation (Shang *et al.* 2014). With this approach, prerequisite knowledge of coding is replaced by the need to execute a detailed trace of an application, where the trace is identified with the main communication and computation activities (Shang *et al.* 2007c). Thus, the main advantage to this approach is that users are not required to have highly detailed knowledge of the program being considered. Consequently, a new application can be assessed with relative ease. However, these programs must have an exposed API that enables a developer to change its inherent behavior. This implies that a parallel application can be developed from serial programs if one knows the APIs in advance.

The use of API-inspired parallelization tools inevitably requires end-users to learn and manually employ the associated APIs within the source code of the serial program. This compromises the usability of these tools owing to the difficulty involved. Furthermore, it has been observed that specialized programs are pervasive, even though most of them do not expose their APIs to end-users (Mackie 2009). Therefore, a well-accepted approach should be able to convert serial programs, regardless of whether their APIs are exposed, into parallel modules without modifying the source code. Such an approach requires a centralized control structure to facilitate the exchange of data between modules and to allocate tasks across computer clusters. Figure 1 shows a schematic of the developed approach.

The PCsP enables modules to be linked with different spatial and temporal model representations. A universal plugin is inserted into the program prior to link-enabling the compiled counterpart. To improve the development environment, functionalities provided in the domain-specific and general-purpose approaches are also adopted, including distributed processing, COM automation, dynamic data exchange, and centralized control. The PCsP does not require users to learn parallel programming APIs and their code. Moreover, it enables even novice users to introduce parallelism into serial programs.

## PCSP APPROACH

### General description

Time-dependent models are generally used to simulate the time evolution process of hydrodynamics. Models of this type are solved using a multi-grid algorithm in combination with a hierarchy of discretization methods, such as the finite difference method and the finite element method (Marczak 2006). Indeed, these models can be solved to a given accuracy by choosing the iterative method on each mesh that satisfies the so-called smoothing property (Ng *et al.* 2009). The idea behind this technique is to accelerate the iterative solution to a discrete set of algebraic equations on a very fine and finite difference/element volume mesh by taking some of the iterations on a coarser mesh. These iterations can, in turn, be accelerated by iterations on an even coarser mesh. The process is repeated until only a very coarse mesh is used at the most basic level. Practically, the solution to each time-dependent model follows a similar pattern. These processes work on a large set of discrete algebraic

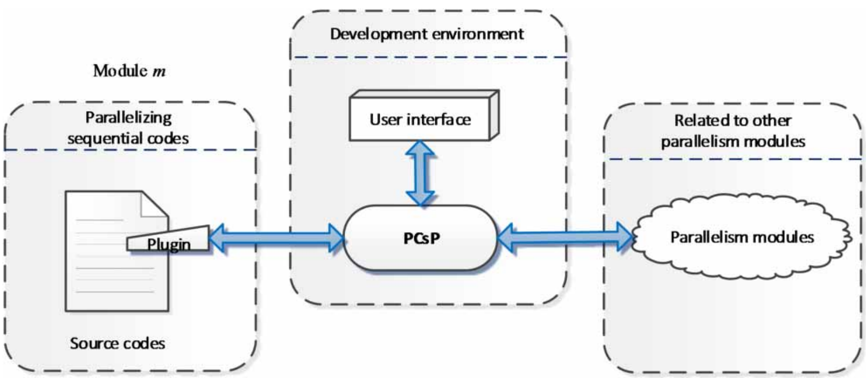


**Figure 1** | Schematic of the development environment for parallel computing.

equations that undergoes repeated cycles in order to create new (and better) solutions to a target problem (Panayirci 2010). Typically, such a set is initialized using the given conditions, and the final solutions are created until a termination criterion (usually the number of function evaluations performed) is met. Each cycle returns a solution-set object for setting boundary conditions for the next cycle until a desired result is ultimately obtained. The general framework for solving a time-dependent model is shown in Figure 2.

In Figure 2, the process for solving a model is structured as a time-ordered series of iterations. For each cycle in the ordering, a new solution set is obtained by solving the discrete set of algebraic equations with a fixed or varying time-step, whereby the solution set from the previous cycle is used to configure the boundary conditions of the next cycle. Based on this typical behavior, time-dependent

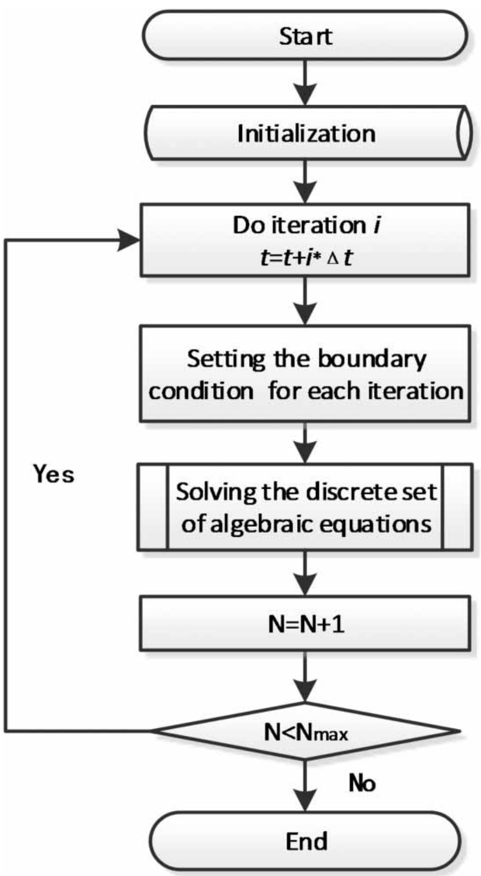


**Figure 2** | General framework for solutions to time-dependent models.

models can be concurrently solved by using the solution-set from one model to change the boundary conditions of another model on a different computer. Moreover, the computation flows of the models can be modified. Thus, the procedure can be concurrently processed by a cluster of computers. Moreover, the dynamic interactions of the physical process can be synchronously simulated using independent mathematical models, provided that these models are controlled at each time-step when exchanging data.

Traditionally, the open modeling interface (OpenMI) is used at run-time to exchange data between models. OpenMI adopts a 'request and reply' mechanism, whereby linked models may run asynchronously with respect to the time-steps. Interfaces are created using C# and Java. They are represented by a set of software interfaces that predefine how the programs are executed and how data are transferred. Models that comply with the interface standard can be configured to exchange data during computation (at run-time). To become an 'OpenMI-compliant' component, a time-dependent model should be programmed in accordance with the OpenMI standards, and it must pass the dimensional checks on the quantities linked.

However, this approach is unsuitable for hydrodynamic simulators mainly because such programs are developed without the intention of linking them to other programs. To overcome this problem, we developed an integration method to simultaneously run these programs by sharing information at each time-step. The method inherits from OpenMI the concept of modular components based on the consideration of data exchange. By providing a centralized control structure, modules can be synchronously controlled. The control structure is shown in Figure 3. With this approach, the modules can form links and share information with each another via the master. In addition, PCsP is committed to interacting with users to fully immerse the user in the virtual environment, such as by showing how the environment immediately responds after the boundary conditions are modified.

The concept discussed above and its equivalents in terms of the PCsP structure, shown in Figure 3, correspond to a server and clients in a client–server model. In this structure, the PCsP plays the role of a master (server) that communicates with each module (client) in a central-access

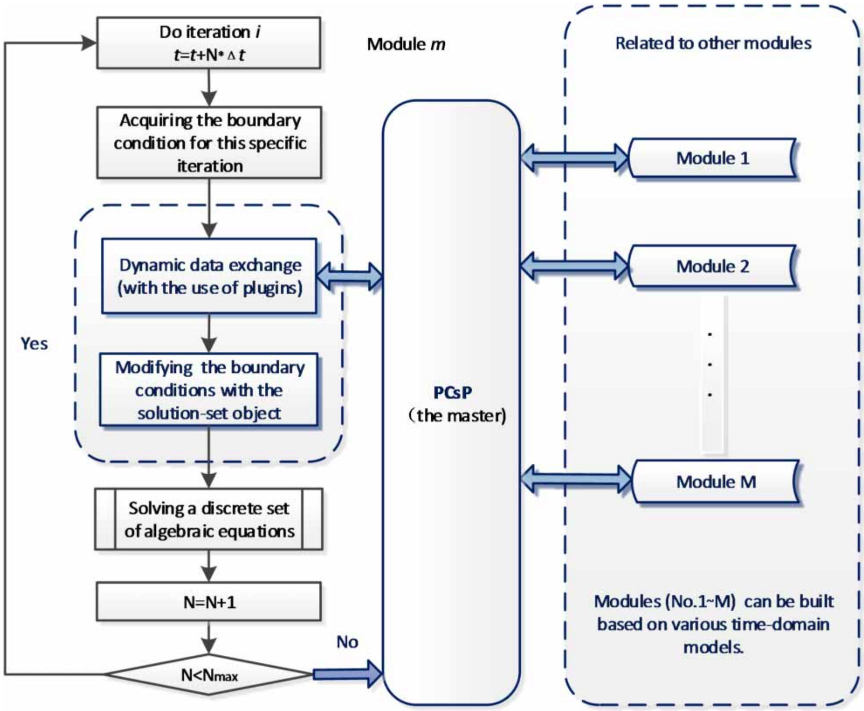


**Figure 3** | Association between the PCsP and parallel modules in a cluster-computing system.

environment via transmissions among themselves. As depicted in Figure 3, by inserting a procedural function (i.e. a plugin) into the loops, a program is converted into a module for parallel application. The module changes its behavior and executes instructions according to the rules determined by the master.

First, the module passes attribute information, such as inputs, outputs, and geometries (i.e. grids) to the master. Then, the master compiles the required information and passes it along to the end-user. Based on the user's objective, the information collected from the other modules is transmitted to the target module for computation. Finally, after the module completes the computation for each time-step, it sends the output results to the master, and the master proceeds with the computation of the next cycle.

## Development of PCsP

Based on the proposed system architecture, the detailed object-design scheme and the proposed solutions for achieving the desired system features are presented in this section. TCP/IP networks (Wang *et al.* 2011b) are adopted for communication among the modules in hydro-informatics platforms. Owing to its interoperability with other modules, the Internet was considered the appropriate solution in our study. The PCsP connects models not only from different suppliers, domains, and concepts, but also with different spatial and temporal resolutions. The PCsP can be described at two levels. At the user level, it provides a set of standard plugins, although modules are allowed to exchange data among themselves on a time-step basis as they run. At the IT level, the PCsP is the engine for the parallel application of interest. The models involved mutually depend on each other's calculation results. Any model in the multi-grid algorithm can be configured without further programming with regard to the data exchanged at run-time. Linked modules can synchronously run with respect to the time-steps, and data represented on different geometries (i.e. grids) can be seamlessly exchanged.

To facilitate data exchange, the input and output (IO) are synthesized in a suitable geometric format; a 'geometries' file is written to facilitate the conversion of IO data. The IO data are written in a single file, while the grid information is preserved in another file. Both files are generated and maintained by the master. The master reads the data and suitably translates them for the input of the target module based on the geometries file in which the IO data are included.

## Links

The data and information flows form a complicated network that involves all the linked modules. Data exchange is achieved within the network. The exchanged data and information can be classified according to the abstracted levels, namely, the respective model, value, and grid level.

*Model level*. Model-level information includes the modules for the application, the attributes of the included modules, the links between modules, and the data flow for each link. We denote the model-level links as shown in Table 1.

Here, $M_i$ represents the $i$th module, and $ML_{i,j}$ denotes the link between source module $M_i$ and target module $M_j$. This is a one-way link. The reverse direction for this link is denoted by $ML_{j,i}$ for unique correspondence.

*Value level*. Value-level links constitute a sub-set of model-level links. They relate the outputs of the source module to the inputs of the target module. A model-level link, $ML_{i,j}$, generally contains multiple value-level links because both the source module and target module are predominantly of the multiple-input multiple-output (MIMO) type. An output of the source module may be sent or converted to multiple inputs of the target module. We map the value-level links as shown in Table 2.

**Table 1** | Model-level links

| | $M_1$ | … | $M_i$ | … | $M_j$ | … | $M_n$ |
|---|---|---|---|---|---|---|---|
| $M_1$ | | … | $ML_{1,i}$ | … | $ML_{1,j}$ | … | $ML_{1,n}$ |
| … | … | | … | … | … | … | … |
| $M_i$ | $ML_{i,1}$ | … | | … | $ML_{i,j}$ | … | $ML_{i,n}$ |
| … | … | … | … | | … | … | … |
| $M_j$ | $ML_{j,1}$ | … | $ML_{j,i}$ | … | | … | $ML_{j,n}$ |
| … | … | … | … | … | … | | … |
| $M_n$ | $ML_{n,1}$ | … | $ML_{n,i}$ | … | $ML_{n,j}$ | … | $ML_{n,n}$ |

**Table 2** | Value-level links

| | $QT_1$ | … | $QT_j$ | … | $QT_n$ |
|---|---|---|---|---|---|
| $QS_1$ | $QL_{1,1}$ | … | $QL_{1,j}$ | … | $QL_{1,n}$ |
| … | … | … | … | … | … |
| $QS_i$ | $QL_{i,1}$ | … | $QL_{i,j}$ | … | $QL_{i,n}$ |
| … | … | … | … | … | … |
| $QS_m$ | $QL_{m,1}$ | … | $QL_{m,j}$ | … | $QL_{m,n}$ |

Here, $QS_i$ represents the $i$th output of the source module, $QT_j$ represents the $j$th input of the target module, and $QL_{i,j}$ represents the link between $QS_i$ and $QT_j$.

*Grid level*. The output of the source module may be converted to suit the input format of the target module. The grid-level information serves as the basis for spatial conversion. Two vectors and a two-dimensional matrix are used to record the linking information at the grid level. One vector stores grid-indexing information with respect to the output data of the source module, while the other vector stores grid-indexing information with respect to the input data of the target module. The relationship between the two vectors is recorded using a two-dimensional matrix. Their correspondence is shown in Table 3.

Here, $GS_i$ represents the $i$th grid of the mesh from where output data are taken and $GT_j$ represents the $j$th grid of the mesh that will receive the input data. The conversion from $GS_i$ to $GT_j$ is denoted by $GL_{i,j}$.

## Data conversion

Data conversion defines how the requested inputs are generated. For the conversion, an internal mapping matrix is required. The mapping matrix is created by specifying and

**Table 3** | Grid-level links

| | $GT_1$ | … | $GT_j$ | … | $GT_n$ |
|---|---|---|---|---|---|
| $GS_1$ | $GL_{1,1}$ | … | $GL_{1,j}$ | … | $GL_{1,n}$ |
| … | … | … | … | … | … |
| $GS_i$ | $GL_{i,1}$ | … | $GL_{i,j}$ | … | $GL_{i,n}$ |
| … | … | … | … | … | … |
| $GS_m$ | $GL_{m,1}$ | … | $GL_{m,j}$ | … | $GL_{m,n}$ |

mapping the links at the link level. This process is performed once the initialization has started. During the mapping process, the source module uses the mapping matrix to perform spatial transformation by multiplying the vector of values associated with the grid of the source modules with the mapping matrix. The PCsP performs spatial transformation using the following operation:

$$t_i = \sum_{j=1}^{n} (a_{ij} s_j + b_{ij}),\ i = 1, 2, \ \ldots, m \tag{1}$$

where $s_j$ is the source-module output, $t_i$ is the input for the target module, $a_{ij}$ is a weight coefficient for the transition from the output of the source module to the input for the target module, and $b_{ij}$ is the adjusting coefficient determined by module calibration. For example, for two modules that adopt different basic levels, the value of the adjustment is the difference of the basic levels selected. Equation (1) can be expressed in matrix form as follows:

$$\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ldots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{12} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ldots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{2}$$

For convenience, the matrices in Equation (2) are consolidated as follows:

$$\mathrm{T} = \mathrm{A} \cdot \mathrm{S} + \mathrm{B} \cdot \mathrm{I}$$

where T is the output vector, S is the input vector, A is the weighting matrix, B is the calibration matrix, and I is the unit vector.

### Time-step control

The master visualizes the relations between modules to facilitate the creation of links. The outputs and inputs are selected based on user needs. Accordingly, the master automatically generates the links between levels. Before the computation process, the master passes the configuration instructions to the modules through an IO data file and a geometries file. At each time-step, a module returns feedback through the plugin in response to the master request, and the master achieves IO translation upon receiving and compiling the feedback. Thus, the linking network is generated, and data and information are freely exchanged over the PCsP. The computational flowchart is shown in Figure 4.

The master determines which modules will be involved. It loads and launches the modules in an orderly manner according to a time-step control strategy. The time-step specifications for different models are found to be variable mainly because they are developed from programs of different model types. Thus, it is necessary to control all the modules so they complete their respective time-step computations within each cycle to ensure the consistency of data operations. For example, if Module A lags behind Module B by a few time-steps, the master will push Module A forward through several consecutive time-steps such that the duration of the computation in progress remains within a small time-step.

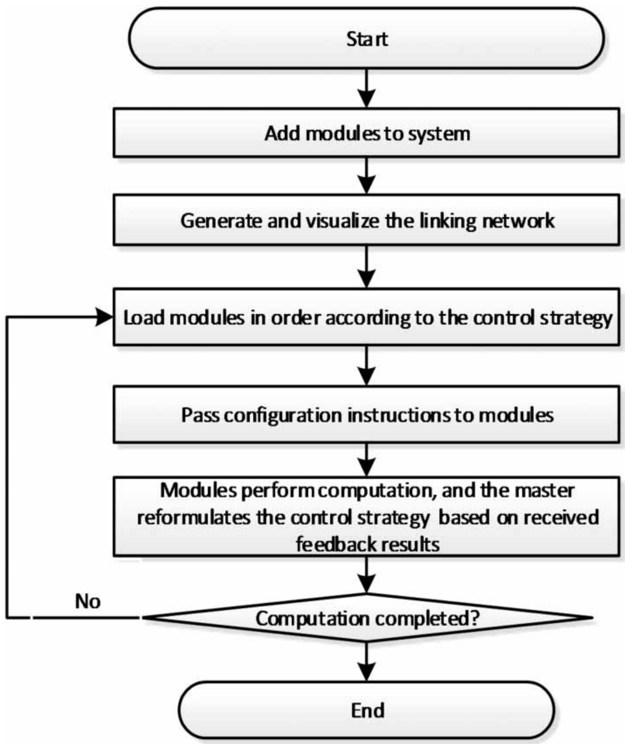


**Figure 4** | Computation flowchart.

Time-stepping simulations produce data roughly in proportion to the number of time-steps computed. Simulations with many steps can result in large data files. To conserve storage, only the computation results from two consecutive time-steps are stored in the master for each module; these steps are needed for the subsequent computation cycle. Then, linear interpolation between the steps can be adopted to ensure that sufficient information is available for smooth animation.

It should be noted that a 'deadlock' occurs when Modules A and B are intertwined. Each module demands from its counterpart the results from the next cycle as the boundary conditions for the current cycle. In these cases, each module requires the results of its counterpart from the next cycle for computation when they are mutually conditional. However, the next cycle will proceed only if the current computational cycle is complete. To decouple the process execution, we use the results from the current cycle as the input for the next cycle.

## Plugins

By adding the plugins, the program becomes a module of the application. Using an example that considers a sequential mathematical model developed in Visual Fortran for Windows operating systems (C++), this section defines the rules of such a plugin for data exchange. Plugin development involves three tasks: creating tables for cross-referencing between different data types, defining the reference style of function calls, and setting the packing and unpacking rules for datasets.

Computing languages generally adopt a different naming style for each data type. Therefore, a reference table is required for data exchange. Table 4 summarizes the cross-referencing between the basic data types in Fortran and C++.

Further, in practice, a Fortran program cannot call a C++ function unless this function has been referenced in the Fortran program. The reference style used in this study is outlined below:

```
extern "C" __declspec(dllexport) void __stdcall FUNNAME(int &param)
```

where 'void' represents the types of values returned by a function. The use of 'void' herein indicates that any data type in Table 4 is allowed as a return value. In addition, 'FUNNAME' represents the name of a function, and 'int & param' represents the option for the parameter term of the function. The reference style for a C++ function defined by a Fortran program is given below:

**Table 4** | Cross-referencing between basic data types in Fortran and C++

| Fortran data type | C++ data type | Fortran data type | C++ data type |
|---|---|---|---|
| INTEGER(1) | char | CHARACTER (1) | unsigned char |
| INTEGER(2) | short | COMPLEX(4) | struct complex4{ |
| INTEGER(4) | int, long | | Float real, image; |
| REAL(4) | float | COMPLEX(8) | struct complex8{ |

```
Interface
FUNCTION FUNNAME(param)
!DEC$ ATTRIBUTES STDCALL, ALIAS :'_FUNNAME @4' :: FUNNAME
INTEGER(4) :: param
END FUNCTION
End interface
```

Finally, different models running in an application will generate mass data of various data types. For convenience, we propose that the data is to be packed in a uniform format. For example, the data in Model_Time_Info can be packed in a dataset as follows:

```
TYPE HYG_MODELTIMEINFO    !Model_Time_Info
INTEGER(4) :: HYG_StructKind    !Data code, as shown in Table 3.1
CHARACTER(LEN=10) :: HYG_ChDateStart,HYG_ChDateStop !Start and ending date(YYYY-MM-DD)
CHARACTER(LEN=8) :: HYG_ChTimeStart,HYG_ChTimeStop !Start and ending time(hh:mm:ss)
INTEGER(4) :: HYG_IStepSec    !Time step in second
END TYPE HYG_MODELTIMEINFO
```

The format of the packed dataset is shown in Figure 5. Here, the number in each block represents the number of bytes in each component. For the dataset package, the first block of four bytes records the total number of bytes in the incoming dataset; the next block of four bytes (HYG_StructKind) records the type of data.

To ensure that the dataset is correctly unpacked, the receiving end unpacks it through the following procedure:

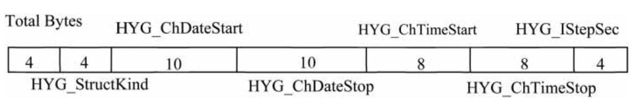


**Figure 5** | Format of the packed dataset.

(1) by reading the first four bytes, the receiving end determines the total number of bytes in the incoming dataset and allocates storage space accordingly; (2) by reading the next four bytes, the receiving end determines the type of received data; (3) based on the type and size of the incoming dataset, the receiving end assign values to the corresponding variables; (4) finally, the receiving end transforms the data using the cross-reference table (e.g. Table 4), when the sending and receiving ends are programmed using different languages.

## Templates

The plugins serve as the middleware for the PCsP. By virtue of the plugins, a component can receive data from any other component of the PCsP. Nevertheless, a minor modification is required in the source model to make it a component of the PCsP. In order to facilitate the use of the PCsP, we developed a suite of programming templates along with a graphical user interface. These templates are presented below for the benefit of other model developers who can copy the templates to the corresponding locations in their model's source code without further modification. Moreover, such locations are easy to find because the copy procedure merely involves the head of the main program, the IO, and the calculation boundary settings.

### Transformation template for a sequential program

```
IF(IFNET==1) THEN   !Component-based flag
If(N==1) Then   !First time-step
Call HYG_InitCtrlEnv !Networking and initialization programming environment
Call HYG_SetModelName('ModelName'C)   !Model name
Call HYG_SetModelDateTime(DStart,TStart,DEnd,TEnd,DT)!Time information
Else
Call OutputResult(N)   !Output calculation results
Endif
Do While(.TRUE.)
CommCode=HYG_WaitComm()   !Wait for control command
If(CommCode==1002)then   !Receive configuration command
Call CreatePoExTable   !Create import and export table
Call HYG_ReplySrv(1002)   !Send model information to the Master
Else If(CommCode==1006)then   !Receive calculation command
Call HYG_ReplySrv(1006)   !Receive import and export tables for this function
Call OutputResult(0)   !Output initial values
Else If(CommCode==1010)then   !Receive further instructions
Call HYG_ReplySrv(1010)   !Receive boundary information
Call GetBound   !Modify boundary conditions based on the data received
Endif
End Do
ENDIF
```

### Template for creating the PCsP's IO table

```
SUBROUTINECreatePoExTable
!First build a table that imports data.
Call HYG_SetPoImTableSize(nTotalPoImQu,nTotalGrid) !Allocate memory for the data imported; the two parameters are total import and total grids.
{Use function "HYG_SetPoImQu(iQuCode,chQuName)" } to set quantity code, name, and relationship.
{Use function "HYG_SetPoImFlag(iQuCode,iGrid,iFlag)" } to set import flag on the appointed grid based on the grid number.
!Build a table that exports the data.
Call HYG_SetPoExTableSize(nTotalPoExQu,nTotalGrid) !Allocate memory to the exporting data; the two parameters are total number of export and total grids.
{Use function "HYG_SetPoExQu(iQuCode,chQuName)} to set quantity code, name, and relationship.
{Use function "HYG_SetPoExFlag(iQuCode,iGrid,iFlag)" } to set export flag on the appointed grid based on the grid number.
END SUBROUTINECreatePoExTable
m
```

### Template for outputting results (OutputResult)

```
SUBROUTINE OutputResult(iStep) !Step i is the current time-step.
DO i=1 HYG_GetTotalExItem()   !Get the total number of export items.
QuCode=HYG_GetExQuCode(i)   !Get the quantity code.
DO k=1,HYG_GetExTotalGridPerQu(i)   !Get the total export grids for the current quantity.
GridCode=HYG_GetExGridNum(i,k) !Get the export grid number.
{Based on the quantity code, grid numbers, and the current time-step, give corresponding data to variableVal.}
Call HYG_SetExData(i,k,Val)   !Save data in the export table.
ENDDO
ENDDO
Call HYG_SendExData   !Send the export data to the control program.
END SUBROUTINE OutputResult
```

### Template for setting the calculation boundaries (GetBound)

```
SUBROUTINEGetBound
nLine=HYG_GetTotalImItem()   !Get total import items.
DO i=1,nLine
iQuCode=HYG_GetImQuCode(i)   !Get import quantity code.
nCol=HYG_GetImTotalGridPerQu(i) !Get the total import grid for the current quantity.
DO J=1,nCol
iGridCode=HYG_GetImGridNum(i,j)   !Get the import grid number.
Val= HYG_GetImData(i,j)   !Get the import data.
{ Modify model boundary conditions based on the quantity code, grid number, and data.}
ENDDO
ENDDO
END SUBROUTINEGetBound
```

Note that the functions with the header 'HYG_' are modularization middleware functions. An illustration for instructions used in the templates is given in Table 5.

**Table 5** | Annotation for the coding of instructions

| Coding | Annotation | Coding | Annotation | Coding | Annotation |
|---|---|---|---|---|---|
| 1000 | Inputs list | 1005 | Output table | 1010 | Push forward a step |
| 1001 | Outputs list | 1006 | Computation instruction | 2000 | Remotely request model information |
| 1002 | Configuration instruction | 1007 | Timing | 2001 | Remotely invoke model |
| 1003 | Model name | 1008 | Input data | 9998 | Computation completed |
| 1004 | Input table | 1009 | Output data | 9999 | Quit computation |

## IMPLEMENTATION ISSUES

This study employed real examples representing different combinations of various sequential models in order to analyze the effectiveness of the PCsP, which can also be extended to more complex situations. The core problem in applying the PCsP is its feasibility and effectiveness with combinations of models. Therefore, this paper highlights the technical details of the coupling of individual models. Further details regarding individual model development can be found in the literature (Ye *et al.* 2013; Ye *et al.* 2014; Guo *et al.* 2015).

### Parallel calculation for sequential processes

In some cases, there is no strict requirement for model coupling. One can directly transfer a time-step value from the source component, N, to the target component and use it as the boundary condition for the target component, $N+1$. All components can be simultaneously calculated. In our example, a river is divided into two types of sections (upstream and downstream) that are modeled using two one-dimensional hydrodynamic models (with the same model engine). Then, the coupling of the two section types is calculated. The results calculated for the entire river are compared.

Suppose there is a river (Figure 6) with 239 sections numbered 1 through 239. The outer boundary conditions are as follows: Upstream Section 1 is given a constant flow rate as the procedural boundary condition, and Downstream Section 239 is given a constant water level as the procedural boundary condition. Then, the data from Sections 1 to 126 are employed to build a model, namely, 'Model A' (these sections are numbered according to their original section numbers). Further, the data from Sections 126 to 239 are used to build another model, namely, 'Model B' (these sections are numbered 1 to 114). To establish a connection between the two models, the flow rate of Section 126 in Model A is matched with that of Section 1 in Model B. Similarly, the water level of Section 1 in Model B is matched with that of Section 126 in Model A.

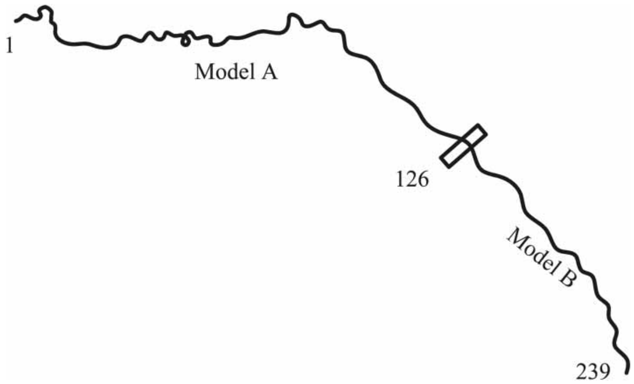


**Figure 6** | Layout for simultaneous calculations using the same sequential program.

To verify the calculation accuracy of the combined model, the calculation results of the combined and original models are compared. The results show that the different water levels for the connecting section (Section 126), as indicated in Figure 7, have a maximum difference of 0.006 m, meaning that it is highly accurate.

### Parallel computation of different types of serial programs

As the study area increases, the efficiency of the sequential process calculation decreases. However, a complicated geological structure can be separated into many sub-areas. In each sub-area, the appropriate grids, models, and model combinations can be employed to approximate the complicated

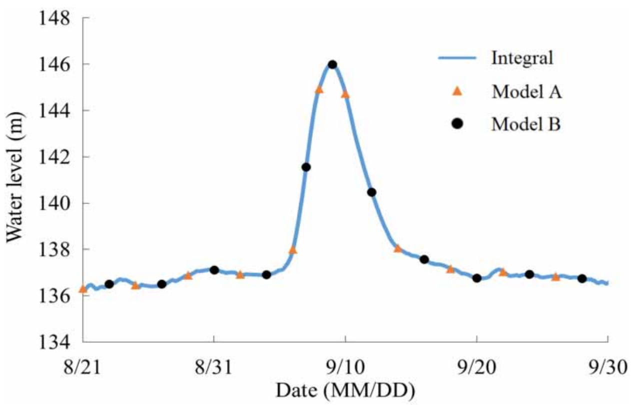


**Figure 7** | Comparing the water level at the connecting section (Section 126).

geological structure. As shown in Figure 8, the upstream region covers an area of $100 \times 1$ km with a river slope of 0.03%. The downstream region is a two-dimensional area of $12 \times 10$ km with the river floor as its base. The height of the two-dimensional area is equal to that of the river mouth, and the outlet is in the middle of one side, with an exit that has a width of 1 km. These two areas meet at the axis. All other sides have solid boundaries.

To test the calculation accuracy of the component-based model, we considered the one-dimensional model, two-dimensional model, and combination of both models (simulating the entire area). We then compared the changes to the water level after coupling the models, as shown in Figure 9. The results of the three models show good agreement.

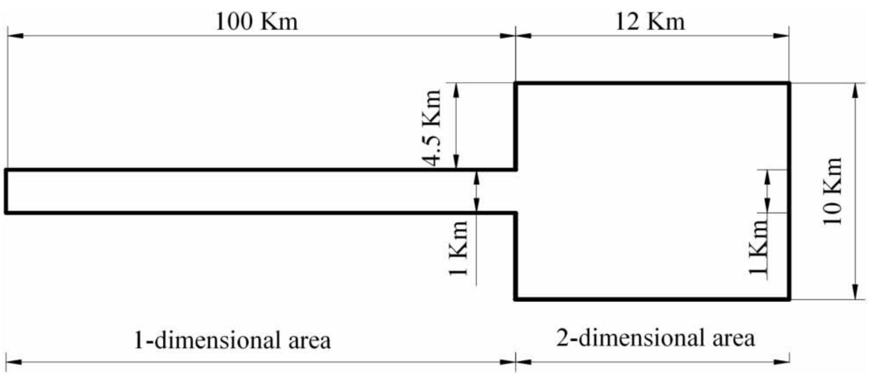


**Figure 8** | Calculation domains for the case study.

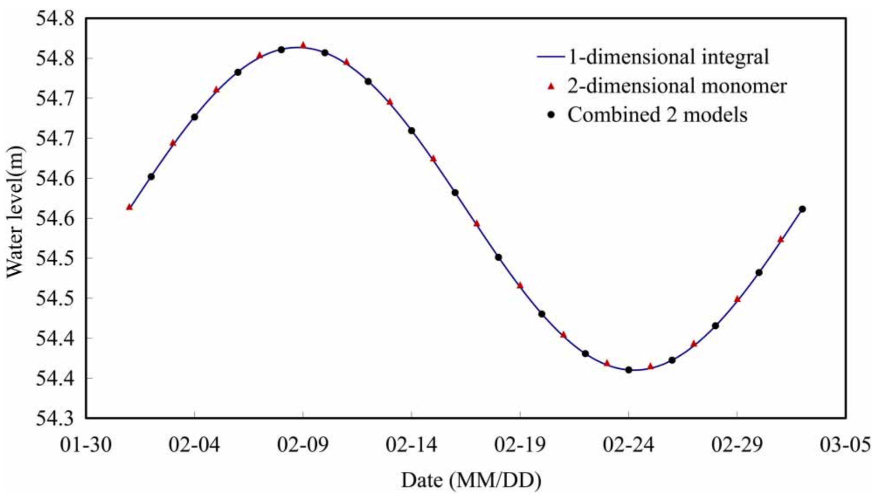


**Figure 9** | Comparison of the water level at the connecting section.

This example demonstrates that the combined model can fully exploit different sub-models in complex situations and that it provides excellent approximation of the original models as a whole. This procedure improves the accuracy and efficiency of model calculation. Moreover, component-based modeling prevents development duplication and offers considerable portability.

## Verifying the efficiency and accuracy of parallelized sequential programs

When using the PCsP, the original model should be partitioned into sub-models, and the sub-models should jointly approximate the original sequential model. The internal boundaries are coupled by combining the models without changing the original model's structure. This solves the problem in terms of the parallel computation of a sequential model. Here, the PCsP is employed to model a heat-conduction process using a sub-area calculation. The efficiency and accuracy of our parallel process are verified by comparing it with a sequential process. The heat-conduction equation is described as follows:

$$\text{Control equation:} \frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0,\ x \in (0, T) \tag{3}$$

$$\text{Initial condition: } u(x, 0) = u^0(x),\ x \in (0, 1) \tag{4}$$

$$\text{Boundary condition: } u(0, t) = u(1, t) = 0,\ \ t \in (0, T) \tag{5}$$

Let the entire computation area be divided into two sub-areas of the same length. Let the first half be denoted as Area ***A*** and the second half be denoted as Area ***B***. To calculate the coupling in the current example, the two sub-areas may extend into each other by length h. Thus, there are three overlapping areas, with an $x$ value in the two sub-areas in the interval (0, 0.5 + h] to [0.5–h, 1). There are 22 grids (i.e. sections). The two sub-areas are modeled using Model A and Model B, as shown in Figure 10.

The efficiency was compared by adjusting the number of sub-areas and performing calculations using the same number of time-steps on the same quad-core personal computer (PC) (2,000 time-steps in this case). The speedup $S$ is

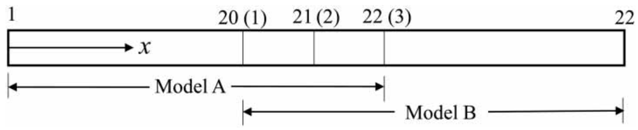


**Figure 10** | Calculation area and partitioning.

defined as $S = T_s/T_p$, where $T_s$ is the execution time of the sequential algorithm, and $T_p$ is the execution time of the parallel algorithm. The speedup value and the sub-area numbers represent the single-zone efficiency. This means that the efficiency is calculated in a single processor (where each sub-area has its own processor). The test results for the sub-areas are presented in Table 6. Table 7 summarizes the differences between the different calculation methods.

The results indicate that the PCsP maintains the calculation accuracy and improves the calculation efficiency. We found that when the optimal number of sub-areas is four, the speedup reaches a maximum value of 2.82. Further, the speedup decreases with the number of the sub-areas, because the calculation capacity of the quad-core PC reaches its maximum level, and additional processes are queued to await execution.

## Practical problems with the Three Gorges project

This project covers a massive area, as shown in Figure 11. The study area has a complex river–lake system. Owing to

**Table 6** | Comparing the calculation time and speedup in sub-areas

| Sub-area N | Calculation time (s) | Speedup *S* | Single-zone efficiency *S/N* |
|---|---|---|---|
| 1 | 2,489 | 1 | 1 |
| 2 | 1,248 | 1.99 | 0.995 |
| 4 | 883 | 2.82 | 0.705 |
| 6 | 996 | 2.50 | 0.417 |

**Table 7** | Calculation results with different calculation approaches ($t = 0.5$)

| X | Actual ($10^{-3}$) | Proposed scheme ($10^{-3}$) | Literature (Zhang & Shen 2002) ($10^{-3}$) | This study ($10^{-3}$) | Diff (%) |
|---|---|---|---|---|---|
| 0.1 | 2.2224 | 2.3654 | 2.3409 | 2.3731 | 6.78 |
| 0.3 | 5.8184 | 6.1928 | 6.1266 | 6.2088 | 6.71 |
| 0.5 | 7.1919 | 7.6547 | 7.5670 | 7.6623 | 6.54 |
| 0.7 | 5.8184 | 6.1928 | 6.1266 | 6.2088 | 6.71 |
| 0.9 | 2.2224 | 2.3654 | 2.3409 | 2.3731 | 6.78 |

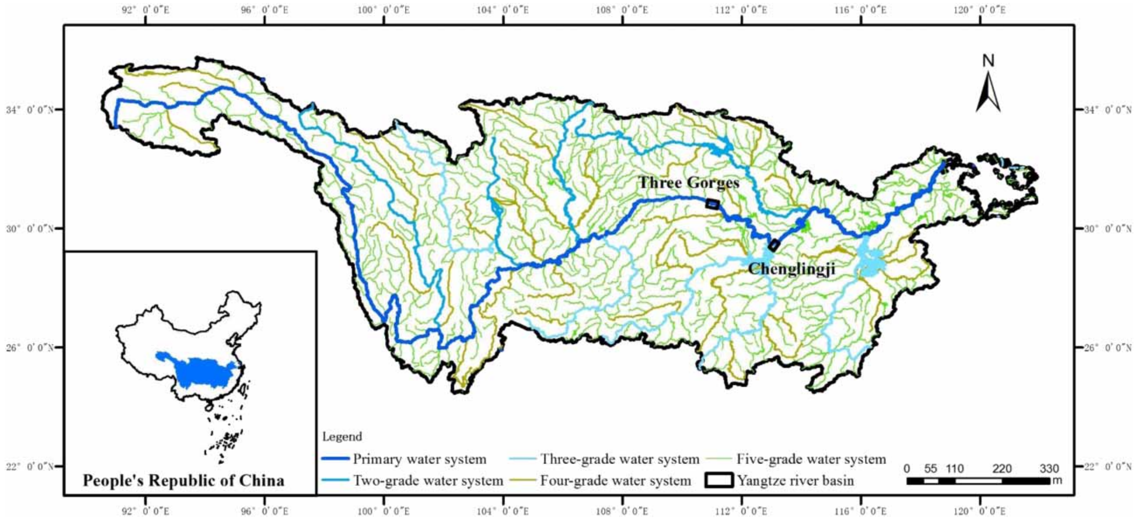


**Figure 11** | Calculation areas for the case study.

the inextricable nature of this system, the sequential study method eventually loses its effectiveness. Although subsequent studies have made numerous improvements to the traditional sequential calculation method (Li *et al.* 2009, 2013; Fang *et al.* 2012), they frequently suffer from unavailability and low efficiency in a river–lake case. Chenglingji is located at the entrance of Dongting Lake, where Lianhuatang is at the connection point. The sequential calculation method is apparently unsuitable for proceeding with a continuous water level computation for Chenglingji.

In this study, two models were employed in the PCsP to evaluate the regulation of the water level at Lianhua Pool before the Three Gorges (TG) Dam (compensation regulation). In addition to high efficiency and accuracy, the PCsP offers significant advantages in terms of simplicity, flexibility, and ease of control and feedback.

Figure 12 shows the water-level variations at the site of the TG Dam. Figure 12(a) shows a flow rate of 60,000 $m^3/s$. However, because the flow rate is too high, it has a relatively minor impact on the compensation regulation at the maximum flood rate. Specifically, it reduces the flood rate of Chenglingji from 65,200 to 63,700 $m^3/s$, and the water level of the TG Dam reaches 153.7 m, which lasts for approximately 3.5 days.

Figure 12(b) shows a flow rate of 55,000 $m^3/s$. The maximum flood rate is reduced to 61,400 $m^3/s$, with the water level of the TG Dam rising to 157.6 m. In this case, the compensation period lasts for 41 days, which effectively controls the flood areas. To reduce the flood rate below the safety level of 60,000 $m^3/s$, as shown in Figure 12(c), the flow rate is reduced further to 50,000 $m^3/s$.

In this case, however, although the maximum flood rate is reduced to 59,200 $m^3/s$, the TG Dam sustains a water level above 160 m for a long period of time. Therefore, flooding occurs despite the regulated compensation. Furthermore, because this phenomenon is influenced by dam drainage, it lasts for 4 days when the flow rate is above 50,000 $m^3/s$. It can be observed that the choice of flow rate has a crucial impact on the regulation of compensation.

According to the Chenglingji compensation regulation, the TG Dam is to be regulated according to the inflow of the section from the TG Dam site to the Chenglingji Project, which includes the Dongting lake system. This serves to control the water level or flow rate at Chenglingji (Lianhuatang) under the threshold value.

At present, the compensation regulation method is subject to the following rules. The TG Dam starts to store water when the flow rate at Chenglingji is greater than a certain value (starting regulation flow rate $Q_1$). The decrease of the flow rate at the TG Dam within each time step shall be in direct proportion to the increase of the flow rate at Chenglingji within the same time step. The proportionality coefficient is called the coefficient of impounding velocity

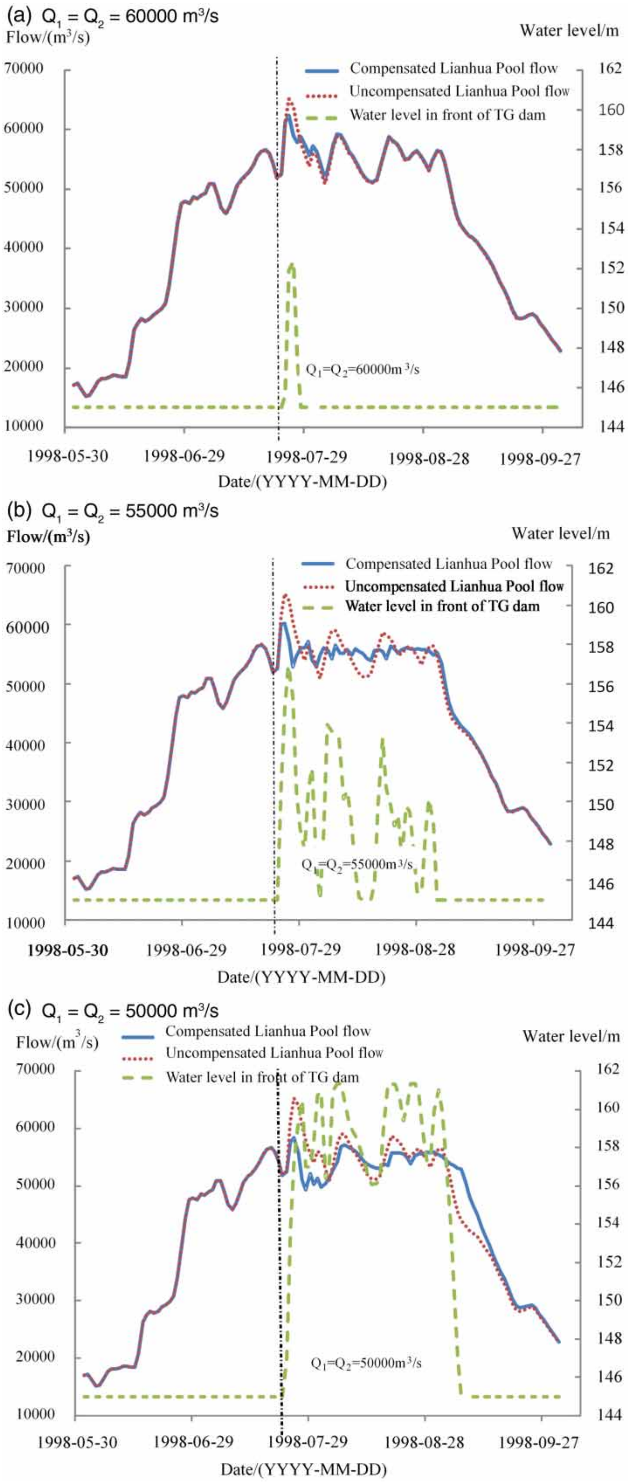


**Figure 12** | Water-level variation at the TG Dam site.

($k_1$), which is set by users. The minimum generating flow shall be no less than 25,000 $m^3$/s, and the TG Dam will be operated by maintaining the water level after the level in front of the Dam reaches 161.3 m. Water drainage shall be increased at the TG Dam, and extra water stored in the earlier stage shall be discharged when the flow rate at Chenglingji is less than a certain value (drainage flow rate $Q_2$, and $Q_1 \geq Q_2$). The water level in front of the dam shall be controlled during the water drainage so that it is lowered at a uniform speed (the coefficient of drainage velocity ($k_2$) is set by users) until it decreases to the flood control level (145 m). To enhance the flexibility of the compensation regulation, regulation parameters may be different values based on the periods of time.

Thus, by studying the Chenglingji flow rate and changes in the water level at the TG Dam, we believe there are huge defects in the regulation method currently adopted, which involves taking the real-time flow rate at Chenglingji as the basis of the compensation regulation. This is because the method fails to consider the time delay for water waves to spread to Chenglingji from the dam. We recommend adoption of the ladder regulation strategy for the flow rate to implement a divided period compensation.

## CONCLUSION

In this paper, a PCsP approach was proposed to address the existing need for reconstructing sequential models to suit high-performance computing platforms. With the proposed approach, a sequential model is easily transformed into a module of the PCsP by merely inserting a procedural function in its main program. The PCsP controls each module through instructions contained in the dataset, where physical quantities and parameters are represented as blocks of data and locations are expressed as grid numbers. Moreover, templates and a graphical user interface were presented to enable novice users to manage the PCsP. An auxiliary function library for simplifying the modeling process was developed using hybrid programming. This method changes neither the calculation logic nor the original model. Moreover, it adopts a simple algorithm for performing distributed and parallel calculations. We tested our proposed platform with real hydrodynamic data. The results indicate that the proposal can successfully exchange data and commands between models while maintaining accurate calculations with high efficiency.

Furthermore, two or more independent models can be combined in an organic unit by a simple operation for studying the complex relations between the models. This widens the scope of application for the PCsP. Dividing and combining the models provides an effective method for addressing complex large-scale problems. Each model can be completed by a different developer, and there is no need for the developers to communicate with each other. Each developer can choose a familiar computational environment for developing the model. When the models are combined, they not only interact through the exchange of data, but they also provide user-defined parameters for data exchange. This enhances the flexibility of the procedure. Thus, the PCsP is suitable for the development of complex mathematical models through collaboration of multiple developers.

In this paper, we furthermore discussed the basic features of the PCsP and its realization. The PCsP provides a novel method for rejuvenating traditional mathematical models by enabling the parallel computing of seamlessly converging models. Nevertheless, the PCsP's features require further improvement and expansion. Future research will focus on two aspects. The first aspect concerns the characteristics of different model combinations. Models can be combined with an explicit or implicit scheme, both of which affect the platform's performance. This paper offers a set of tools designed to couple models on a trial-and-error basis. However, it does not develop clear theoretical conclusions and questions remain. What kinds of models can be combined together and under which scheme? What about the stability and precision of the platform? Can the combined models simulate the physical processes of complex phenomena? The above questions will be covered in future studies on the characteristics of model combinations, which will be an important research contribution to the parallelized model-development approach.

The second focus for future research concerns compatibility. This paper discusses the PCsP's application for environmental modeling and software. However, the PCsP is a generic method for process simulations; in theory, it can be applied to all time-stepping mathematical models. Therefore, PCsP is fully open and applicable to multiple domains and models that simulate the time process and comply with the program structure of time-steps. By exploring the compatibility of the PCsP with such models, we can expand its scope of application. For example, OpenMI is an interface standard based on modular development, and it is widely used to construct many excellent models. Thus, we intend to generate time-series dynamic simulations based on the OpenMI components and the PCsP's data exchange and timing control.

## ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China (Grants 51579248, 51109112, 51309254), and the Open Research Fund of the State Key Laboratory of Hydroscience and Engineering (Grant 2015-B-03), State Key Laboratory of Water Resources and Hydropower Engineering Science (Grant 2014SWG03), State Key Laboratory of Simulation and Regulation of Water Cycle in River Basin (Grant IWHR-SKL-201517), and CRSRI (Grant CKWV2014224/KY). Furthermore, a special fund was received for platform development from the China Institute of Water Resources and Hydropower Research (Contract WR0145B02201500000).

## REFERENCES

Afshari, S., Mandle, R. & Li, S.-G. 2008 Hierarchical patch dynamics modeling of near-well dynamics in complex regional groundwater systems. *J. Hydrol. Eng.* **13** (9), 894–904.

Bugaets, A. N. 2014 Using the OpenMI standard for developing integrated systems of hydrological modeling. *Russ. Meteorol. Hydrol.* **39** (7), 498–506.

Chen, H.-M. & Lin, Y.-C. 2008 Web-FEM: an internet-based finite-element analysis framework with 3D graphics and parallel computing environment. *Adv. Eng. Software* **39** (1), 55–68.

Chui, C.-K., Wang, Z., Zhang, J., Ong, J. S.-K., Bian, L., Teo, J. C.-M., Yan, C.-H., Ong, S.-H., Wang, S.-C., Wong, H.-K. & Teoh, S.-H. 2009 A component-oriented software toolkit for patient-specific finite element model generation. *Adv. Eng. Software* **40** (3), 184–192.

Fang, H., Han, D., He, G. & Chen, M. 2012 Flood management selections for the Yangtze River Midstream after the Three Gorges project operation. *J. Hydrol.* **432**, 1–11.

Fialko, S. 2010 PARFES: a method for solving finite element linear equations on multi-core computers. *Adv. Eng. Software* **41** (12), 1256–1265.

Gregersen, J. B., Gijsbers, P. J. A. & Westen, S. J. P. 2007 OpenMI: open modeling interface. *J. Hydroinform.* **3** (9), 175–191.

Guo, Y., Tang, X., Chen, X. & Shang, Y. 2015 Application of 2-D LES in river flow simulation. *Port Waterway Eng.* **503** (5), 112–116.

Holmes, D. W., Williams, J. R. & Tilke, P. 2011 A framework for parallel computational physics algorithms on multi-core: SPH in parallel. *Adv. Eng. Software* **42** (11), 999–1008.

Jagers, H. R. A. 2010 A Linking Data, Models and Tools: An overview. In: *2010 International Congress on Environmental Modelling and Software: Modelling for Environment's Sake*, Fifth Biennial Meeting, Ottawa, Canada.

Li, J., Marino, M. A., Ji, C. & Zhang, Y. 2009 Mathematical models of Inter-plant economical operation of a cascade hydropower system in electricity market. *Water Resour. Manage.* **23** (10), 2003–2013.

Li, F., Shoemaker, C. A., Wei, J. & Fu, X. 2013 Estimating maximal annual energy given heterogeneous hydropower generating units with application to the Three Gorges system. *J. Water Resour. Plan. Manage.* **139** (3), 265–276.

Mackie, R. I. 2009 Design and deployment of distributed numerical applications using. NET and component oriented programming. *Adv. Eng. Software* **40** (8), 665–674.

Marczak, R. J. 2006 Object-oriented numerical integration – a template scheme for FEM and BEM applications. *Adv. Eng. Software* **37** (3), 172–183.

Márquez, A. L., Gil, C., Bas, R. & Gómez, J. 2011 Parallelism on multicore processors using parallel. FX. *Adv. Eng. Software* **42** (5), 259–265.

Moore, R. V. & Tindall, C. I. 2005 An overview of the open modeling interface and environment (the OpenMI). *Environ. Sci. Pol.* **8** (3), 279–286.

Ng, S. M. Y., Wai, O. W. H., Li, Y.-S., Li, Z.-L. & Jiang, Y. 2009 Integration of a GIS and a complex three-dimensional hydrodynamic, sediment and heavy metal transport numerical model. *Adv. Eng. Software* **40** (6), 391–401.

Panayirci, H. M. 2010 Efficient solution for Galerkin-based polynomial chaos expansion systems. *Adv. Eng. Software* **41** (12), 1277–1286.

Shang, Y., Jin, Y. & Wu, B. 2007a Fault-tolerant mechanism of the distributed cluster computers. *Tsinghua Sci. Technol.* **12** (Suppl. 1), 186–191.

Shang, Y., Wu, B., Li, T. & Fang, S. 2007b Fault-tolerant technique in the cluster computation of the digital watershed model. *Tsinghua Sci. Technol.* **12** (Suppl. 1), 162–168.

Shang, L., Wang, Z., Zhou, X., Huang, X. & Cheng, Y. 2007c TM-DG: A trust model based on computer users' daily behavior for desktop grid platform. In: *Proceedings of the 2007 Symposium on Component and Framework Technology in High-performance and Scientific Computing*, HPC-GECO/CompFrame 07, October 21–25, Montréal, Québec, Canada, pp. 59–66.

Shang, Y., Liu, R., Li, T., Zhang, C. & Wang, G. 2011a Transient flow control for an artificial open channel based on finite difference method. *Sci. China Technol. Sci.* **54** (4), 781–792.

Shang, Y., Lu, G., Shang, L. & Wang, G. 2011b Parallel processing on block-based Gauss–Jordon Algorithm for desktop grid. *Comput. Sci. Inform. Syst.* **8** (3), 739–759.

Shang, Y., Shang, L., Lu, G., Ye, Y. & Jia, D. 2014 Using the high-level based program interface to facilitate the large scale scientific computing. *Scientific World J.* **2014**, Article ID 914514.

Wang, H., Fu, X., Wang, G., Li, T. & Gao, J. 2011a A common parallel computing framework for modeling hydrological processes of river basins. *Parallel Comput.* **37** (6), 302–315.

Wang, L., Kunze, M., Tao, J. & von Laszewski, G. 2011b Towards building a cloud for scientific applications. *Adv. Eng. Software* **42** (9), 714–722.

Ye, Y., Hei, P., Liang, L., Jiang, Y., Jia, D. & Shang, Y. 2013 Unsteady flow mathematical model of river basin water allocation. *J. Basic Sci. Eng.* **21** (5), 866–880.

Ye, Y., Liang, L., Zhang, G., Shang, Y., Wang, J. & Jiang, Y. 2014 Numerical simulation of dam-break water flow with complex boundary based on governing equations modification. *J. Hydroelectric Eng.* **33** (5), 99–107.

Zhang, B. & Shen, W. 2002 Note on finite difference domain decomposition algorithm for the solution of heat equation. *J. Numer. Meth. Comput. Appl.* **2**, 81–90.

First received 12 October 2015; accepted in revised form 14 March 2016. Available online 8 April 2016